IN THE UNITED STATED PATENT AND TRADEMARK OFFICE

APPLICATION FOR U.S. LETTERS PATENT

Title:

A FAST ALGORITHM TO EXTRACT FLAT INFORMATION FROM HIERARCHICAL NETLISTS

Inventor:

John W. Regnier

Thomas J. D'Amico
DICKSTEIN SHAPIRO MORIN &
OSHINSKY LLP
2101 L Street, NW
Washington, DC 20037-1526
(202) 828-2232

TITLE OF INVENTION:

A FAST ALGORITHM TO EXTRACT FLAT INFORMATION FROM HIERARCHICAL NETLISTS

FIELD OF THE INVENTION

[0001] This invention relates generally to the field of design and testing of semiconductor integrated circuits and more particularly to error detection and testing of semiconductor chip schematic designs used in semiconductor manufacturing.

BACKGROUND OF THE INVENTION

[0002] Computer aided design (CAD) systems are used to design complex integrated circuits or dies. Schematic designs are created using such CAD systems which describe the integrated circuit components and interconnections between components which will be fabricated within integrated circuit dies.

[0003] CAD systems typically store the circuit designs in memory within a hierarchical form or a flat form. The hierarchical structure of a circuit design description is represented by parent-child relationships. The hierarchical representational system breaks the system being designed into a top level module or modules which include smaller blocks within them that make up the top level module. The overall system is the top most module. The top level module does not usually include any reference to gates or other circuits, but rather refers more to the function accomplished by the top or macro level component(s) which are the largest functional component(s) in the design.

[0004] Each next lower level of module includes a smaller number of modules until the level or subset of the total system is reached which contains only primitives or the most basic circuit sub-components, such as a NMOS transistor or a resistor. A hierarchical representation of a circuit design allows designers to solve complex circuit design problems using the divide and conquer approach. Complexity is reduced to the point where the overall design can be understood without being obscured by the details of a very large circuit design. High levels of abstraction permits a designer to understand the system under development at many levels.

[0005] Hierarchical designs can be organized in a number of ways such as "top down" and "bottom up". The top down methodology starts by defining the top level blocks and identifying the sub-blocks necessary to built the top level blocks and further subdivides these sub-blocks into leaf cells, which are the lowest level of a circuit design. For example, a leaf cell may be a cell within a hierarchical representation that does not contain any other cells, thus it is the lowest level in the hierarchical representation. The subdivision of lower components into lower cells or modules usually stops when there is no further significant advantage in reusability and comprehensibility of detail.

[0006] The bottom-up design methodology approach begins from the leaf cells, which are the terminal end of a branch of cells in a tree type circuit hierarchy and thus represent the "bottom" of the tree schematic. Designers begin to construct the higher cell blocks in the circuit design hierarchy using available leaf cells. A designer continues building the next higher blocks until the final design is realized.

[0007] In practice neither bottom-up or top-down approach is used alone. A combination of both approaches is typically used. Usually when decomposing a hierarchy, the design is partitioned into several sub-modules based on their functionality. This is sometimes known as vertical partitioning.

[0008] The second type of CAD system design relationship uses a flat representation of circuit components in a design. Each component connects directly to another component, rather than being defined in a modular approach seen in the hierarchical system, where modules connect to other modules.

[0009] Once a schematic circuit design has been created using CAD systems, it is output as a netlist. Netlists are computer files which contain a textual or character based description of the integrated circuits and the relationships between the circuits or cells that comprise a device described by the netlist or schematic design.

[0010] Netlists can be organized in the hierarchical or the flat form. A flat data netlist contains multiple copies of the circuit modules without boundary descriptions found in hierarchical representations, usually consisting of module or cell instance names. For example, a flat data netlist will list one or more flat paths describing a string of components that are connected at a highest level in the circuit design hierarchy through a lowest component. In other words, a flat path it is a path statement from a starting place, usually the highest point in that particular path, to a specified endpoint marker or either the lowest or bottom-most primitive component in a particular current or circuit path. The start or stop points can also be selected based upon a desire to test a segment within a larger circuit path.

[0011] CAD tools are commonly linked together and must be able to convert from one format to another format depending on the stage of the integrated circuit design process. Hierarchical netlists are typically used when designing the integrated circuit as that format helps designers easily understand how the different components in the system being designed work together. Flat data netlists are usually required when other CAD tools are used such as timing verification, circuit simulation and circuit placement and routing tools.

[0012] One type of electronic design automation (EDA) system used to evaluate and perform error checking on the netlist after the design is compiled includes electrical rule check (ERC) programs. ERC programs look at a netlist schematic and make sure markers or other specific components are present within the schematic design and then evaluates how the system design works with respect to the markers placed in the netlist. The ERC system focuses on looking for a particular electrical marker/component such as high voltage (HV) net markers, super voltage (SV) net markers, pad markers, super voltage fuse markers, negative voltage net markers for lower voltage charge pumps (lower than zero volts) and any other type of marker.

[0013] ERC systems require a flat data netlist representation to perform rule checking in some cases. (Some checks are done solely on a hierarchical representation, such as verification that a particular marker is at all levels of the hierarchy) ERC systems which must scan through each instance of a cell or combination of cells in a large integrated circuit design would require very large amounts of processing time to perform ERC functional checks on hierarchical lists. Consequently, a hierarchical netlist representation must be converted to a flat data netlist to decrease ERC processing time. An ERC system starts with a beginning reference point and then performs rule checks for circuit

components that are listed from the reference point along the absolute path statement. A reference point may be a cell description, net descriptor (an electrical connection, e.g. "wire" between components or cells), HV marker or any specified circuit component or group of circuit components and/or markers and/or net descriptors. The ERC system evaluates how a marker or circuit component interacts with the rest of the design by looking for failure modes that occur when certain combinations are present, such as improper associations of a component with a marker e.g. pad marker associated with a circuit component which is incompatible with the pad. Markers (also called flat data search targets with respect to flat data search programs) are used by ERC programs to identify circuit components within a hierarchical netlist which are being evaluated for improper combinations with other circuit components during ERC checks.

[0014] The flat data netlist used in ERC systems includes an absolute path down to the marker via instance names. Flat net information includes a path statement for every marker along with net names without listing cell instance identifiers in the path description usually from a top level to a lower level cell or component in a circuit schematic which lie on a net. Cell instances are required to describe a hierarchy of larger to smaller cells.

[0015] Conventional flat data programs perform searches in a hierarchical netlist to assemble flat path statements. Such a program typically starts at a top level cell then looks at every instance in that cell progressing or traversing down through each subsequent level in the hierarchical circuit schematic or netlist until the flat data search system has assembled a flat description for the netlist which includes all the markers throughout the entire system. Such flat data programs assembles path statements as the program progresses from a top descriptor to a bottom descriptor. This can take a great deal of time when dealing

with very large systems such as a DRAM. For example, a DRAM can have hundreds of millions of cells which the flat list search system must go through and create an absolute path statement to populate the flat data file. It is not uncommon for a conventional flatlist system to take three for four days to go through a DRAM circuit to assemble the flat data with flat path statement for each net running from one specified point to another specified point. Thus, a need exists to improve the ability for flat data programs to assemble flat data required for rapid development of integrated circuits.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus which assembles absolute path statements in flat data by scanning cells and circuit components of a hierarchical netlist, recording the path statement from boundary of the scanned hierarchical cell to a cell or marker within the cell, then appending the recorded path to the path of a subsequently scanned hierarchical cell or component which contains the previously record cell thus avoiding the necessity of rescanning previously encountered cells.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features and advantages of the invention will be better understood from the following detailed description which is provided in connection with the accompanying drawings.

Figure 1 shows a schematic describing a plurality of hierarchical elements describing a circuit;

Figure 2 shows a computer system employing an embodiment of the invention;

Figure 3 shows exemplary system architecture for one embodiment of the invention;

Figure 4 shows exemplary data structures employed by one embodiment of the invention;

Figure 5 shows a processing sequence for processing and testing a netlist and assembling an absolute path statement constructed in accordance with an exemplary embodiment the invention;

Figure 6 shows output from one embodiment of the invention; and

Figure 6a shows additional detail of the figure 6 output in accordance with an exemplary embodiment of the invention.

DESCRIPTION OF THE INVENTION:

[0016] In an exemplary embodiment the invention rapidly extracts flat data from the bottom up, storing flat path statements for the components it has scanned, then combining or prepending the stored flat path statement segment to the flat path data sequence being assembled when identical element instances are subsequently encountered. In other words, the first time a cell is encountered, it is scanned and flat data is stored within a data structure for holding cell information. When the flat data engine progresses to another cell in the cell hierarchy, it checks to see if it has previously encountered the newly scanned cell

and recorded the cell's path data. If the cell has been scanned and flat data recorded, then the recorded flat data path segment is added to the path that is being assembled at that point in the traversal thereby avoiding the need to repetitively scan and record each element each time another cell is encountered. Thus, no cell or module will need to be searched more than once during the traversal. This is a major speed enhancement for flat data extraction within a hierarchical netlist with multiple instances of identical modules.

[0017] Figure 1 shows an exemplary set of hierarchical schematics having several top level modules which are scanned by the invention. It should be noted that Fig. 1 is only an exemplary hierarchical schematic and that the invention has universal application to any electrical circuit represented in a hierarchical form. As mentioned above, the highest level in a hierarchical design is the top level which usually describes the largest module(s) or component(s) which make up the hierarchical design. Cells which contain other cells, for example ercFlatC (I0) 31, are also known as parent cells as a parent cell contains one or more child cells.

[0018] Sub-modules or components contained within the top-most or higher levels module or component schematics are referred to as child cells. The child cells of Figure 1 include ercFlat A (I0) 35, ercFlat A (I1) 41 and ercFlatB (I2) 51 since they are within ercFlatC (I0) 31. Child cells can have either specific circuit components within them or other child cells within them or both.

[0019] A child cell which has no further child cells within it is also known as a leaf cell and is the lowest form of hierarchical circuit description. Referring to Fig. 1, instance 2 (I2) of ercFlat A 11, instance 3 (I3) of ercFlatB 21, instance 0 (I0) of ercFlat A 35,

instance 1 (I1) of ercFlat A 41 and instance 2 (I2) of ercFlatB 51 are all leaf cells since no further cells are contained within those cells. The term "instance" is used to distinguish between cells with the same name that are used within a circuit representation, for example instance zero of ercFlatA, instance one of ercFlatA and instance two of ercFlatA. On the other hand, instance 0 (I0) of ercFlatC 31 contains ercFlat A (I0) 35, ercFlat A (I1) 41 and ercFlatB (I2) 51 thus it is not a leaf cell. Note that ercFlatC is a child cell as ercFlatC 31 is contained within a larger hierarchical cell, ercFlatTOP 2. Leaf cells can contain a description of the components within the cell or the leaf cell can merely contain a description of the function the cell performs.

[0020] The term instances, such as instance zero (I0) of ercFlatC 31 in Fig. 1, is a descriptor that symbolizes an occurrence of a cell in a specific circuit design rather than a generic description of the cell in isolation and without reference to the cell's location within a circuit design's hierarchy. The same generic cell can have several instances or occurrences of itself. For example, the generic ercFlat A in Fig. 1 has three instances, ercFlat A I0 35, ercFlatA I1 41 and ercFlatA I2 11. Each occurrence of a cell within a hierarchical design is given an instance number, e.g. within ercFlatTOP 2, there are three cell instances I0 (ercFlatC 31), I1 (ercFlatB 11) and I2 (ercFlatA 11). Within ercFlatC 31 there are multiple cell instances including I0 (ercFlatA 35), I1 (ercFlatA 41) and I2 (ercFlatB 51).

[0021] The schematic of Fig. 1 also describes nets which are used within a hierarchical circuit design. A net is a descriptor that references an interconnection of some type between cells or components found within a hierarchy. Nets are found at the top-most hierarchy of a circuit design running to parent and child cells. For example, referring to

Fig. 1, the TOP_1 net 3 runs to ercFlatA I0 11 where it terminates. Within ercFlatA I2 11, another net, net INA_1 13, runs from an incoming cell hierarchy boundary point (CHBP) to a component within the cell ercFlatA I2 11 and terminates at an interconnection point with yet another net, net 2 17, at another CHBP. A signal or voltage goes through components in a cell, but CAD tools see different net names for carrying a voltage from one component to another component in most cases. A net terminates at a component then another net begins on the other side of the component. Thus, cells or nets define beginning and ending interconnections or interfaces between cells and from other nets running to points within cells.

[0022] An absolute path statement refers to a hierarchical description which indicates a string of elements in terms of cell or circuit component instance names within a hierarchical circuit design which lie between two points. For example I0 / I0 / R0 describes an absolute path to the R0 resistor within ercFlatA I0 (35) in the Fig. 1 schematic. The term "absolute" is a more specific definition of hierarchical path. An absolute path statement takes you exactly to one place. Thus, multiple hierarchical net paths can exist that describe the same absolute net path. For example, an absolute path to the INA_1 net in Fig. 1 would be I0/I0/INA_1, which refers to the unique net identifier INA_1 inside instance zero of ercFlatC and instance zero of ercFlatA.

[0023] In contrast to an absolute path, there are several hierarchical paths that could be used to describe net INA_1 in instance zero of ercFlatA that is within instance zero of ercFlatC. Other hierarchical paths to INA_1 include the /TOP_2 net in Fig. 1 that is a hierarchical path that describes the same net path as an absolute path to INA_1 within ercFlatA within ercFlatC. Another hierarchical path, I0/INC_1, also describes another

hierarchical path to INA_1 within ercFlatA which is within ercFlatC. Thus, when desiring to define a component by a path statement, a unique absolute path statement is used rather than a hierarchical path statement.

[0024] A hierarchical path statement describes cells within cells using cell names and circuit component descriptions, for example, ERC flat C / ERC flat A / R0. This is the hierarchical path to the R0 resistor. This path statement does not indicate each and every instance name (e.g. I0, I1, I2). Instead, the path statement only indicates that if you have a Cell C, there will be a Cell A in it and a resistor R0 in it. A hierarchical path statement uses generic description rather than a reference to the exact cell with the exact instance names for the multiple instances within the larger cell e.g. first instance of cell C, second instance of cell C and third instance of cell C.

[0025] Referring to Fig. 2, a computer system 71 employing an embodiment of the invention is disclosed. A storage device/medium 73 stores and retrieves the flat data extraction program as well as the variables and other required data elements. Command inputs are made through input devices 75 to activate the flat data extraction program and input commands. The screen system 79 can display status, commands and results associated with the flat data extraction program. The processor/memory 77 executes commands and manipulates data associated with the flat data extraction program. The printer 81 outputs flat data as specified in the flat data extraction program or as directed by commands input through input devices 75. Output flat information or data can also be stored into a data storage medium or transferred to other computer systems.

[0026] Referring to Fig. 3, exemplary data structures employed by one embodiment of the invention are disclosed. A hierarchical netlist 1 contains the data which will be parsed. Parsed data 89 includes cells data (%CELL) 91, current cell being scanned (%currcell) and instance data (%INST) 101. Marker data 102 is stored for local nets and flat net paths including the %HV data structure 103 which includes local HV marker data and %FlatHV data structure 107 which stores the flat data which is being assembled. The flat data cache data structures 108 stores flat path data, cell segment information and traversal information. The %HVCache data structure 109 includes cell instance data which is used to append to upper level flat data sequences when a previously scanned cell instance is encountered during netlist traversal. The %SEEN data structure 111 stores instance recognition information which is used to identify which cell instances have been encountered during netlist traversal.

[0027] Referring to Fig. 4, major components of a system in one embodiment of the invention are disclosed. A hierarchical netlist 1 is input into the flat data extraction engine 151 for processing. The ReadNetlist module 153 reads in a hierarchical netlist file 1. The ParseNetlist module 155 parses the hierarchical netlist 1, identifies cell and instance information then copies cell and instance information into variables %CELLS 91 and %INST 93 and then returns the name of the last cell which should be in the top level cell 99 within the netlist 1 hierarchy, which in this case is ercFlatTOP 2. The getLocalHV module 157 creates a list of local HV nets and markers within each instance type and stores this data in the %HV data structure 103. The getFlatHV module 159 finds all flat HV nets and markers in the input netlist 1. The newNetMap module 161 updates net mapping from parent to child cells. The addPreEach module 163 adds strings to the beginning of

each element in the flat data 167 which is being created. Several output modules exist 165 for printing or outputting data such as flat data 167. The term module is used here to describe a computer processing sequence, function, subfunction, member function, method, subroutine or section of a subroutine.

[0028] Referring to Fig. 5, one exemplary processing sequence that may be used to carry out the invention is disclosed. Processing is initiated by loading a hierarchical netlist 1 into the processing system 71 through either the input device 75 or from storage 73 and the ReadNetlist module 153 reads in hierarchical netlist 1 at processing segment 200. Next, the input hierarchical netlist 1 is parsed by the parseNetlist module 155 storing cell data into variable %CELL 91, instance data into %INST 93 at processing segment 201. In processing segment 203, the getLocalHV module 157 parses the instances (%INST 101) finds local HV nets/markers 103 for each individual cell type (i.e. searches one instance of Fig. 1 ercFlatA 11, ercFlatB 21, ercFlatC 31) and stores local marker and cell data into data structure %HV 103. It should be noted that processing segment 203 could also be performed later in the processing sequence such as, for example, after processing segment 219 since segment 221 and 223 use the cell and HV marker. In processing segment 205, the getFlatHV module 159 is called and the top level HV nets/markers 15 are added to the %FLATHV 107 data structure. Next, the getFlatHV module 159 checks to see if there is another instance in the cell which is currently being scanned at processing segment 209. The getFlatHV module 159 will then get the identity or name of the next unscanned child instance in the current cell being scanned at processing segment 211. Note that the next child instance may be a cell instance, a marker or other component such as a resistor. The GetFlatHV module 159 will then determine if the next child instance is a cell at

processing segment 211. If the next child instance is not a cell, then processing is branched back to processing segment 207 and then GetFlatHV 159 will again check to see if there is another instance in the current cell being scanned at processing segment 209. If there is another instance, then GetFlatHV 159 will again check to see if the scanned instance is a cell at processing segment 213. This loop from processing segment 213 back to 207 will continue until either there are no other instances (processing segment 209) or the instance being scanned is a cell (processing segment 213).

[0029] If the getFlatHV module 159 determines there is another cell instance within the current cell hierarchy at processing segment 213 (e.g. in the Fig. 1 schematic, there are multiple child cells within ercFlatC 31), then the getFlatHV module 159 will then determine the net mapping for the child cell from the child's parent to the child by calling the netNetMap module 161 at processing segment 215 which will store net mapping information. The getFlatHV module 159 will then determine if the child cell has been scanned previously at processing segment 217 by determining if the selected cell's instance name is stored in the SEEN data structure 111. If the child cell has not been scanned, then getFlatHV module 159 will mark the cell as visited in the SEEN 111 data structure at processing segment 219. In processing segment 221, the local child cell flat data net and marker entries will be added to the %FlatHV data structure 107. Next, the local child cell flat data net, marker and component entries will be added to the %FlatHVcache 109 data structure at processing segment 223. The %FlatHVcache 109 data structure holds instance data structures which are retrieved at subsequent processing segments for addition to upper level hierarchy flat data stored during a netlist traversal sequence leading up to the currently selected cell instance. At processing segment 225, the getFlatHV module 159 calls itself

recursively thereby returning to processing segment 207 with the currently designated child cell as a top cell or traversal starting point.

[0030] Processing segments 209 through 213 will be repeated parsing through non-cell elements until another child instance cell is encountered, assuming additional instances are within the currently selected cell. At processing sequence 215, the getFlatHV module 159 will again perform mapping for child cell from the child's parent to the child. Next, theGetFlatHV module 159 will determine if it has previously encountered the selected cell instance by reference to the SEEN data structure 111. If getFlatHV 159 finds the instance identifier which matches the currently selected cell in the SEEN 111 data structure, then getFlatHV 159 will retrieve the matching instance's flat data segment from %FlatHVcache data structure 109 and append the matching flat data segment to the sequence of higher level flat data corresponding to the hierarchical netlist path currently being traversed stored in the %FlatHV data structure 107. Next, at processing segment 233 the getFlatHV module 159 will copy child and parent cell relationship data back into the %FlatHVcache 109 data structure and return to processing segment 207 to continue processing.

[0031] When the getFlatHV module 159 determines at processing segment 209 that there are no additional instances along the hierarchical traversal path being processed, then getFlatHV 159 will either terminate completely at processing segment 235 or the current getFlatHV 159 module will terminate and return to the getFlatHV module 159 that previously called it at processing segment 227. Again, child cell data will be copied back to parent cell data elements within the %FlatHVcache data structure 109 to facilitate further processing and tracking of the current point of the netlist traversal. In such a manner, the progressively called getFlatHV 159 modules which executed a copy of itself at

processing segment 225 will begin to complete processing and reverse the netlist traversal to the cell instance which is above the current cell which is being mapped.

The processing sequence of 207 to 213 will be re-executed until another child cell instance is encountered. The processing sequence from 214 to 233 will be re-executed when a traversal selects a cell instance which matches an instance identifier in the SEEN 111 data structure, at which point the getFlatHV module 159 will retrieve the matching flat data and append the flat data to the higher level flat data corresponding to the currently netlist traversal sequence path. The higher level flat data is thus hierarchically and traversally interconnected with the appended data through the traversal sequence. In this manner, instance by instance parsing, mapping and storing is avoided saving considerable time. Complete mapping will still have to be accomplished for cells which have not yet been encountered during traversal of the netlist in processing segments 217 through 225. Processing is complete when all recursively called modules have terminated and there are no further instances at processing segment 235.

[0032] Referring to Figure 6, output for an exemplary embodiment of the invention is shown. The first set of entries 301 under the %HV header correspond to step 203 data. The second set of entries 303 under the %FLATHV header correspond to the final flat data which is being created in the %FlatHV data structure 107 as flat data segments (and their corresponding highest level absolute net paths) are assembled during traversal. The third set of entries 305 under the %FLATHVCACHE header correspond to the data being accumulated in the %FlatHVcache data structure 109.

[0033] Referring to Fig. 6a and also to the hierarchical representation in Fig. 1, flat data output under the %FLATHV header 303 is explained. The first set of entries 401 indicate

the top path keys on the left which defines the upper hierarchical point of the flat data path sequence. The entries on the right of the dashes 403 indicate the flat data following the top net identifier on the left to a particular flat path search target point at the end of the traversal path, a HV marker in this case 417. Referring to Fig. 6a, the first element "/" 405 indicates the top cell which is being scanned, in this case ercFlatTop IO 2. Top_2 407 refers to the top net starting point which is being mapped. The "/X0" entry 411 refers to the first search marker, a HV marker in this case, which was encountered while traversing along the top_2 net (Fig. 1, 5). Next, the "/I0" entry 413 refers to the ercFlatC I0 cell in figure 1 (Fig. 1, 31). Next, the "/I0" entry 415 refers to instance zero of ercFlatA (fig. 1, 35) along the traversed net. Next, the "/X0" entry 417 refers to the final search target along the traversed net, in this case a HV marker within ercFlatA (Fig. 1, 35). In this manner, flat data is described from any number of top nets to a search target element.

[0034] It is also possible to include any number of element descriptors in the flat data path if it is so desired. Alternative embodiments can also change the starting point for the flat data segment mapping to an element or cell other than a top net. Other nets can be specified as a starting point. Terminal points as well as intervening flat data which is stored or ignored can also be varied as well.

[0035] An embodiment of the system can also mask flat data, search targets or circuit attributes which generate invalid or undesirable error listings. In this embodiment, a hierarchical netlist 1 is loaded into the processing system 71 and the flat data is extracted as described in Fig. 5 and is stored in the %FlatHV data structure 107. The %FlatHV data structure 107 is input into an electrical rule check (ERC) program module and then error checks are run on the %FlatHV data 107. Error output data from the ERC system is

stored into an %ERCerror data structure which includes an error identifier, a flat path data which corresponds to the source of the error, a cell path associated with the error and a text description of the error. Error listings are then output from the ERC program and then examined by a user.

[0036] If a user determines that the ERC program is generating invalid or undesirable circuit design error listings, the user will then identify the flat path data, error, marker or circuit attribute which is triggering the errors. The user will then input or modify the flat path data, error, marker or circuit data into the processing system 71 as mask input data into a %maskData data structure through the input device 75. The mask input data will then be used by a mask processor module to identify the flat path data which triggers the ERC program to generate the undesirable or invalid ERC error listings. The mask processor module can identify the flat path data associated with a circuit design error as every error output by the ERC system has a unique instance flat path data associated with the circuit design fault or error. In other words, each circuit design error list item is unique because of the unique flat path data which was extracted by the flat data path engine 151 and stored in %FlatHV data structure 107. The mask module will then match the flat path data in the %FlatHV data structure 107 with the mask data, then filter or extract the error listing stored in the %ERCerror data structure as it is output to screen system 79, printer 81 or to a storage medium thereby ensuring the ERC system is producing valid circuit design fault or error listings.

[0037] An embodiment of the invention can also include a flat path data mask sub-module which is incorporated into the getFlatHV module 159. Invalid or undesirable error listings are identified as discussed above, then a user can input mask data into a

%maskData data structure. The getFlatHV module 159 can screen or mask specified flat path data segments as well as markers, circuit attributes or circuit components in the hierarchical netlist data input 1 as it is traversing the netlist and when netlist traversal is completed. The getFlatHV module 159 can compare the data in the %maskData data structure to a flat path data segment which is to be stored in either the %FlatHVCache 109 or the %FlatHV data structure 107, then either discard the flat data segment or proceed to store or append the data segment into the %FlatHV data structure 107. At the end of processing as described in Fig. 5, the getFlatHV module 159 can compare all the flat data stored in the %FlatHV data structure 107 with the data stored in the %maskData data structure and then delete entries which match the mask criteria stored in %maskData data structure. The filtered flat data in %FlatHV data structure 107 is then input into the ERC system which produces a revised circuit design error or fault listing.

[0038] Alternative embodiments of the invention can include the capability to search for any type of markers besides HV markers which are contained in the netlist such as super voltage (SV) net markers, pad markers, super voltage fuse markers, negative voltage net markers for lower voltage charge pumps (lower than zero volts) and any other type of marker, property, attribute assigned to a net, element, cell or instance. Any circuit element can be used as the search target or terminal point of the accumulated flat data generated by the flat data extraction program. Absolute path statement can be created to each search target by use of the search, recording of partial path statements for a cell or primitive contents data (with respect to search target) and the assembly, appending or prepending features of the invention with different specified start and endpoints. A number of steps can be accomplished in different sequences such as searching of each cell type for marker

content and recording the cell type name (first hash table key) and search target contents in the form of an array in the Fig. 4 embodiment. The hash table keys can also be arranged differently or a different form of organization or program may also be used to implement the invention such as Perl, Lisp, C, C++, C# or any other programming language capable of performing functions of the present invention such as storing the required data, organizing the cell and marker data, performing searches, assembly of segments, appending or prepending path fragments.